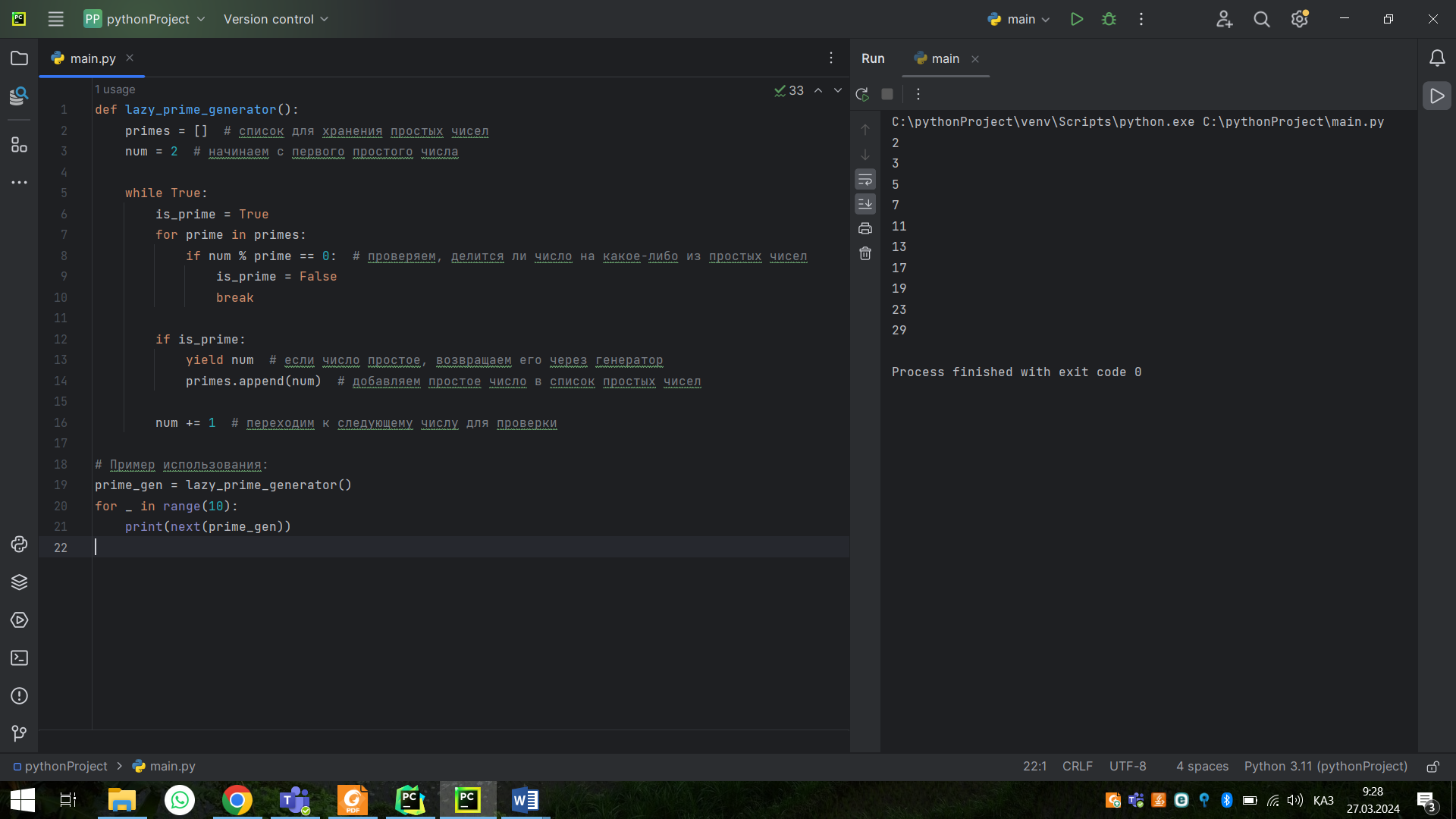
2 вариант

1. Ленивое Вычисление Простых Чисел

- Создать генератор, который лениво генерирует простые числа.



def lazy\_prime\_generator():  
 primes = [] # список для хранения простых чисел  
 num = 2 # начинаем с первого простого числа  
  
 while True:  
 is\_prime = True  
 for prime in primes:  
 if num % prime == 0: # проверяем, делится ли число на какое-либо из простых чисел  
 is\_prime = False  
 break  
  
 if is\_prime:  
 yield num # если число простое, возвращаем его через генератор  
 primes.append(num) # добавляем простое число в список простых чисел  
  
 num += 1 # переходим к следующему числу для проверки  
  
# Пример использования:  
prime\_gen = lazy\_prime\_generator()  
for \_ in range(10):  
 print(next(prime\_gen))

# Вопросы для Подготовки:

1. Что понимается под ленивыми вычислениями в Python и как они реализуются?
   * Цель: Убедиться, что студент понимает концепцию ленивых вычислений и способы их реализации в Python.
2. Каковы преимущества ленивых вычислений?
   * Цель: Проверить, осознает ли студент преимущества использования ленивых вычислений, включая эффективность использования памяти и возможность работы с бесконечными последовательностями.
3. Можете ли вы объяснить разницу между строгими и ленивыми вычислениями?
   * Цель: Оценить понимание студентом различий между строгими (немедленными) и ленивыми (отложенными) вычислениями.
4. Какие сценарии использования ленивых вычислений вы можете привести?

- Цель: Понимание студентами практических применений ленивых вычислений.

1. Как ленивые вычисления могут повлиять на производительность программы?
   * Цель: Оценка способности студента анализировать влияние ленивых вычислений на производительность.
2. Можете ли вы объяснить, что такое генераторы в Python и как они связаны с ленивыми вычислениями?

- Цель: Проверка понимания студентом механизма генераторов и их отношения к ленивым вычислениям.

1. Как можно реализовать функциональные структуры данных в Python? Приведите примеры.

- Цель: Понимание студентами реализации и использования функциональных структур данных, таких как неизменяемые списки или деревья.

1. Что такое мемоизация и как она применяется в контексте ленивых вычислений?
   * Цель: Убедиться, что студенты осведомлены о технике мемоизации и её применении для оптимизации ленивых вычислений.
2. Какие недостатки могут быть у ленивых вычислений?
   * Цель: Оценить критическое мышление студентов и осознание потенциальных ограничений ленивых вычислений.
3. Приведите пример реальной задачи, где вы бы применили ленивые вычисления.

- Цель: Проверить способность студента применять теоретические знания на практике.

Ответы

1. \*\*Что понимается под ленивыми вычислениями в Python и как они реализуются?\*\*

Ленивые вычисления в Python означают отложенное выполнение вычислений до тех пор, пока результаты не станут необходимы. Это реализуется через различные механизмы, такие как генераторы, отложенные вычисления и функции высшего порядка.

2. \*\*Каковы преимущества ленивых вычислений?\*\*

Преимущества ленивых вычислений включают эффективное использование памяти, возможность работы с бесконечными последовательностями данных, улучшение производительности за счёт отложенного выполнения вычислений и возможность использования потоковой обработки данных.

3. \*\*Можете ли вы объяснить разницу между строгими и ленивыми вычислениями?\*\*

Строгие (немедленные) вычисления выполняются сразу, а ленивые (отложенные) вычисления выполняются только при необходимости. В строгих вычислениях все аргументы вычисляются перед вызовом функции, в то время как в ленивых вычислениях аргументы вычисляются только тогда, когда они действительно нужны.

4. \*\*Какие сценарии использования ленивых вычислений вы можете привести?\*\*

Ленивые вычисления могут использоваться для обработки больших объёмов данных, работы с бесконечными последовательностями, оптимизации производительности программы, потоковой обработки данных, а также для создания ленивых структур данных, таких как генераторы и потоки.

5. \*\*Как ленивые вычисления могут повлиять на производительность программы?\*\*

Ленивые вычисления могут повысить производительность программы за счёт отложенного выполнения вычислений, что позволяет избежать излишних операций и использовать память более эффективно. Однако некорректное использование ленивых вычислений может привести к накладным расходам на управление состоянием вычислений.

6. \*\*Можете ли вы объяснить, что такое генераторы в Python и как они связаны с ленивыми вычислениями?\*\*

Генераторы в Python - это специальный тип функций, которые используются для ленивого создания последовательностей данных. Они связаны с ленивыми вычислениями, потому что генераторы вычисляют значения по мере необходимости, что делает их эффективными для работы с большими объёмами данных.

7. \*\*Как можно реализовать функциональные структуры данных в Python? Приведите примеры.\*\*

Функциональные структуры данных в Python могут быть реализованы с использованием неизменяемых типов данных, таких как кортежи, и функций высшего порядка, таких как `map`, `filter` и `reduce`. Примеры включают неизменяемые списки, неизменяемые словари и деревья.

8. \*\*Что такое мемоизация и как она применяется в контексте ленивых вычислений?\*\*

Мемоизация - это техника оптимизации, которая заключается в кэшировании результатов выполнения функции для избежания повторных вычислений. В контексте ленивых вычислений мемоизация может использоваться для кэширования промежуточных результатов вычислений, что может улучшить производительность программы.

9. \*\*Какие недостатки могут быть у ленивых вычислений?\*\*

Недостатки ленивых вычислений могут включать сложность отладки программ, возможные накладные расходы на управление состоянием вычислений, а также неожиданное потребление памяти из-за накопления отложенных вычислений.

10. \*\*Приведите пример реальной задачи, где вы бы применили ленивые вычисления.\*\*

Например, обработка больших объёмов данных, где данные не могут быть загружены в память целиком, или создание бесконечных последовательностей, таких как последовательности Фибоначчи, где результаты вычислений требуется генерировать по мере необходимости.